JOURNAL OF COMPUTER AND SYSTEM SCIENCES 9, 177–195 (1974)

Translatability of Flowcharts into While Programs

TAKUMI KASAI

Research Institute for Mathematical Sciences,
Kyoto University, Kitashirakawa, Kyoto, Japan

Received April 11, 1973

A "while program" [Z. Manna, "Introduction to Mathematical Theory of Computations," to appear] is a simple abstract model of a "GOTO-less program." Some flowcharts, however, can not be translated into while programs in a certain sense [D. E. Knuth and R. W. Floyd, Notes on avoiding GOTO statements, *Information Processing Letters* 1 (1971), 23–31]. That is, there exists a flowchart which is not "congruent" (or computation-sequence-equivalent) to any while program. In this paper we give a necessary and sufficient condition for translatability of flowcharts into while programs.

1. INTRODUCTION

Recently, many authors have pointed out that the use of "GOTO" statements is undesirable, and there have been many attempts to find systematic ways for eliminating GOTO statements [3, 5, 6, 10]. Cooper [4] and Bruno and Steiglitz [1] have shown that any program can be translated into a program having no explicit GOTO statement, by introducing new variables which represent a history of control flow. In this paper a stronger notion of translatability is studied. We consider only translations which do not introduce new variables nor change the sequence of computations and tests.

Now, a flowchart S can be regarded as a special sort of transition graph. We denote by $T(S)$ the set of strings accepted by S. Thus each element of $T(S)$ is a finite sequence of assignment statements and tests which is spelled out by a path from the START node to a HALT node. We say that two flowcharts S_1 and S_2 are congruent if $T(S_1) = T(S_2)$, and a flowchart S is translatable into a while program if there exists a while program α such that $T(S) = T(\alpha)$. Knuth and Floyd [10] have given an example of a flowchart which is not translatable in this stronger sense.

In this paper the notion of "modularity" is introduced, and it is shown that every modular flowchart is translatable into a while program. Furthermore, if a flowchart S is "minimal," then S is translatable if and only if S is modular. Since there is an algorithm for constructing a minimal flowchart (i.e., a flowchart having the fewest nodes), this yields an algorithm for determining if an arbitrary flowchart is translatable.

Copyright © 1974 by Academic Press, Inc.
All rights of reproduction in any form reserved.

Hecht and Ullman [7] have introduced the notion of "collapsibility," and have shown that any D-chart (a flowchart constructed from a while program or a "block form program" in a natural way) is collapsible. In this paper we introduce a somewhat stronger notion of collapsibility, called regular collapsibility, and show it equivalent to modularity.

2. Preliminaries

In this section we recall some well-known concepts about flowcharts and while programs necessary for an understanding of the present paper.

Definition. Let $F = \{f, g, \ldots\}$ and $P = \{p, q, \ldots\}$ be countably infinite sets of symbols. Elements of F are called "function symbols" (or "assignment statements"). Elements of P are called "relation symbols" (or "logical expressions"). For each p in P, let $\bar{p}$ be an abstract symbol (the "negation" of p), and let $\hat{P} = P \cup \{\bar{p} \mid p \text{ in } P\}$.

Definition. A "flow graph" is a 4-tuple $S = (V, \Gamma, \varphi, \text{¢})$, where

(1) V is a finite nonempty set (of "nodes");

(2) Γ is a partial function of V into $V \cup V \times V$. Let a be an element of V.

  (i) If $\Gamma(a)$ is undefined, then a is called a "halt node."

  (ii) If $\Gamma(a)$ is in V, then a is called a "function node."

  (iii) If $\Gamma(a)$ is in $V \times V$, then a is called a "test node."

(3) φ: $V \to F \cup P$ is a partial function called "labeling," such that the following conditions are satisfied.

  (i) $\varphi(a)$ is undefined if a is a halt node,

  (ii) $\varphi(a)$ is in F if a is a function node, and

  (iii) $\varphi(a)$ is in P if a is a test node.

(4) ¢, the "start node," is a distinguished element in V.

Definition. Let $S = (V, \Gamma, \varphi, \text{¢})$ be a flow graph. Let $\tilde{S}$ be the subset of $V \times F \times V \cup V \times \hat{P} \times V$ defined by

$$\tilde{S} = \{(a, \varphi(a), \Gamma(a)) \mid \Gamma(a) \text{ in } V\} \cup \{(a, p, b), (a, \bar{p}, c) \mid \Gamma(a) = (b, c), \varphi(a) = p\}.$$

Elements of $\tilde{S}$ are called "edges." For an edge $e = (a, l, b)$, the node a is called its "initial node," and b its "terminal node." We say b is a "successor" of a. If $a \neq b$,

then b is a "proper successor" of a. Note that there may be two distinct edges connecting the same initial and terminal nodes.

An edge whose initial node is a is said to be "incident out from" a. If W is a given set of nodes, we say that an edge (a, l, b) is incident out from W if a is in W.

A "path" u in a flow graph $S = (V, \Gamma, \varphi, \not\subset)$ is a finite sequence of edges

$$(a_1, l_1, b_1)(a_2, l_2, b_2) \cdots (a_k, l_k, b_k)$$

in S with the property that $b_i = a_{i+1}$ for $1 \leqslant i \leqslant k - 1$. u is also called a "path from" a_1 "to" b_k. If $k = 0$, then u is the "null path," and u will be considered to be a path from any node to itself. The string $l_1 l_2 \cdots l_k$ is called the "trace" of u. $l_1 l_2 \cdots l_k$ is said to be "spelled out" by u. The null path spells out λ (the null string). If W is a given set of nodes, we say that u "passes through" W if $\{a_1, \ldots, a_k, b_k\} \cap W \neq \varnothing$. A path is "elementary" if it does not pass through the same node twice.

Two edges e_1 and e_2 are said to be "associated" (alternatively, e_1 is said to be the "associate" of e_2) if

(1) they are distinct, and

(2) they have the same initial node.

When $\Gamma(a) = (b, c)$, we sometimes write $b = \Gamma_1(a)$ and $c = \Gamma_2(a)$. Thus, for each test node a, $(a, \varphi(a), \Gamma_1(a))$ and $(a, \overline{\varphi(a)}, \Gamma_2(a))$ are associated.

DEFINITION. A "flowchart" is a flow graph $S = (V, \Gamma, \varphi, \not\subset)$ with the following properties.

(1) There is exactly one halt node; and

(2) Every node in V is on a path from the start node to the halt node.

If a flowchart S consists of a single node, then S is said to be "null." Note that this node is the start node as well as the halt node. A null flowchart is denoted by Λ.

The "trace set" of a flowchart S, denoted by $T(S)$, is the set of all strings x such that there exists a path from the start node to the halt node that spells out x. It is apparent that the trace set of an arbitrary flowchart forms a regular set.

Two flowcharts S_1 and S_2 are "congruent," written $S_1 \equiv S_2$, if $T(S_1) = T(S_2)$.

EXAMPLE. Let us consider the flowchart S of Fig. 1(a). The node labeled by a "$\not\subset$" sign is the start node, and the node labeled by a "\$" sign is the halt node. The trace set of S is

$$T(S) = f(gpf \cup g\bar{p}\bar{q})^* g\bar{p}\bar{q}.$$

DEFINITION. The class of "while programs" [11, 10, 8] is defined recursively as follows:

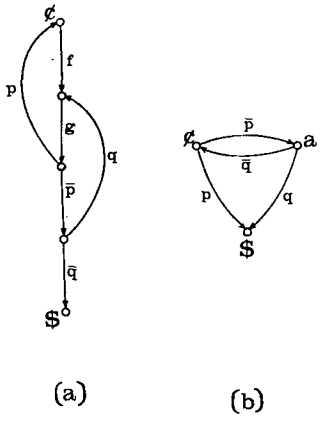
(1) λ is a while program;

(2) each function symbol is a while program;

(3) if α and β are while programs and q is in $\hat{P}$, then

$$\alpha\beta,\ (\textit{if } q \textit{ then } \alpha \textit{ else } \beta),\quad (\textit{while } q \textit{ do } \alpha)$$

are while programs.



(a) (b)

FIG. 1. Examples of flowcharts.

DEFINITION. For each while program α, the trace set $T(\alpha)$ of α is defined recursively as follows:

(1) $T(\lambda) = \lambda$.

(2) $T(f) = f$ for each f in F.

(3) If α and β are while programs and q is in $\hat{P}$, then

(i) $T(\alpha\beta) = T(\alpha)\,T(\beta)$,

(ii) $T(\textit{if } q \textit{ then } \alpha \textit{ else } \beta) = qT(\alpha) \cup \bar{q}T(\beta)$,

(iii) $T(\textit{while } q \textit{ do } \alpha) = (qT(\alpha))^*\bar{q}$,

where $\bar{\bar{p}} = p$ for each p in P.

DEFINITION. A flowchart S is said to be "translatable" if there exists a while program α such that $T(S) = T(\alpha)$.

EXAMPLE. Let S be the flowchart of Fig. 1(a), and let

$$\alpha = fg(\textit{while } p \textit{ do } fg)(\textit{while } q \textit{ do } g(\textit{while } p \textit{ do } fg))$$

Then, $T(S) = T(\alpha)$. Thus S is translatable.

3. Modularity

In this section we introduce the notion of modularity and show that any modular flowchart is translatable.

Definition. Let S be a flowchart, and let \$ be the halt node of S. Let W be a nonempty set of nodes. A path

$$(a_0, l_1, a_1)(a_1, l_2, a_2) \cdots (a_{k-1}, l_k, a_k)$$

is called a "W-path" if $k \neq 0$ and each a_i is in W. W is said to be a "section" of S if, for every ordered pair of nodes a, b in W, there is a W-path from a to b. A section is said to be "maximal" if it is not a proper subset of any section of S.

An edge (c, l, d) is an "exit" of a section W if

(1) c is in W, and

(2) there is a path from d to \$ which does not pass through W. (Thus d is not in W.)

It follows that any section has at least one exit. A flowchart S is said to be "modular" if every section of S has exactly one exit.

Example. The flowchart of Fig. 1(a) is modular. On the other hand, the flowchart of Fig. 1(b) is not modular, since the section $\{¢, a\}$ has two distinct edges $(¢, p, \$)$ and $(a, q, \$)$.

Lemma 3.1. *If (c, l, d) is an exit of a section W, then c is a test node. Furthermore, if (c, l', d') is the associate of (c, l, d), then d' is in W.*

Proof. Since d is not in W, any W-path from c to c must start with the edge of the form (c, l', d'), d' in W. Note that every W-path is nonull.

Definition, Let $S = (V, \Gamma, \varphi, ¢)$ and $Q = (V', \Gamma', \varphi', ¢')$ be flowcharts. If $\tilde{Q} \subset \tilde{S}$, then Q is said to be a "subflowchart" of S. That is, Q is a subflowchart of S if and only if $V' \subset V$, $\Gamma' = \Gamma \mid V_\$'$ and $\varphi' = \varphi \mid V_\$'$, where $V_\$' = V' - \{\text{the halt node of } Q\}$.

Definition. Let $S = (V, \Gamma, \varphi, ¢)$ be a flowchart and let \$ be the halt node of S. A pair (a, b) of nodes is said to be "linked" if every path from a to \$ passes through the node b.

Let (a, b) be a linked pair of nodes. We construct a flowchart S_{ab} as follows.

Let $S_{ab} = (V', \Gamma'', \varphi', a)$, where

(1) V' is the set of all nodes c in V such that c is on a path from a to b which does not pass through b more than once. (Note that if c is in $V' - \{b\}$, then $c \neq \$$ and every successor of c is also in V'.);

(2) $\Gamma' = \Gamma \mid V_{\$}'$ and $\varphi' = \varphi \mid V_{\$}'$, where $V_{\$}' = V' - \{b\}$.

Clearly, S_{ab} is a flowchart (thus a subflowchart of S). S_{ab} will be called the "subflowchart determined by (a, b)". Note that b is the halt node of S_{ab}.

LEMMA 3.2. *Let W be a maximal section of a modular flowchart S. Let $(a, \bar{q}, b)$ be the exit of W and (a, q, c) be the associate of $(a, \bar{q}, b)$. Then both (a, b) and (c, a) are linked. Furthermore, if $S_{ab} = (V', \Gamma', \varphi', a)$ is the subflowchart determined by (a, b), then $V' = W \cup \{b\}$. (See Fig. 2.)*

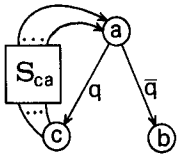


FIG. 2. Subflowchart S_{ab}, where S_{ca} includes both c and a.

Proof. We first show that (a, b) is linked. Let $(a_0, l_1, a_1)(a_1, l_2, a_2) \cdots (a_{k-1}, l_k, a_k)$ be a path from a to the halt node \$. Let a_i be the first node on this path which is not in W (since \$ is not in W, such a node always exists). Suppose that a_j is in W for some j, $i < j$. Then $W \cup \{a_i, a_{i+1}, \ldots, a_j\}$ is a section of S. This contradicts the maximality of W. Thus a_j is not in W for $i \leqslant j \leqslant k$. Hence (a_{i-1}, l_i, a_i) is the exit of W. Thus $a_i = b$, so that (a, b) is linked. An analogous argument shows that the pair (c, a) is linked.

Let $S_{ab} = (V', \Gamma', \varphi' a)$ be the subflowchart determined by (a, b). We now show that $V' = W \cup \{b\}$. Let d be in W. Then there exist W-paths u from a to d and v from d to a. Then $uv(a, q, b)$ is a path from a to b which does not pass through b twice. Thus d is in V'. Hence $V' \supset W \cup \{b\}$. To see the reverse inclusion, consider a path

$$(a_0, l_1, a_1)(a_1, l_2, a_2) \cdots (a_{k-1}, l_k, a_k)$$

with $a_0 = a$, $a_k = b$ and $a_i \neq b$ for all i, $0 \leqslant i < k$. It suffices to show that $\{a_0, a_1, \ldots, a_{k-1}\} \subset W$. Clearly, a_0 is in W. Suppose that $\{a_0, \ldots, a_{i-1}\} \subset W$ and a_i is not in W. Then (a_{i-1}, l_i, a_i) must be the exit of W. Thus $a_i = b$, so that $i = k$. Hence $V' = W \cup \{b\}$.

THEOREM 3.3. *Every modular flowchart is translatable.*

Proof. Let $S = (V, \Gamma, \varphi, \not{c})$ be a modular flowchart, and let \$ be the halt node of S.

We will construct a while program α such that $T(S) = T(\alpha)$. If there is no section in S, then α can be made up by means of only (*if–then–else*) and concatenation.

Suppose that S has n sections, $n \geqslant 1$, and that the theorem is true for all flowcharts with fewer than n sections. Let W be a maximal section, and let $(a, \bar{q}, b)$ be the exit of W. Let (a, q, c) be the associate of $(a, \bar{q}, b)$. By Lemma 3.2, both (a, b) and (c, a) are linked. Let S_{ab} and S_{ca} be the subflowcharts determined by (a, b) and (c, a), respectively (see Fig. 2). Clearly, S_{ca} is modular. Thus, by the induction hypothesis, there exists a while program α_1 such that $T(S_{ca}) = T(\alpha_1)$. Let

$$\alpha_2 = (\textit{while } q \textit{ do } \alpha_1).$$

Then,

$$\begin{aligned} T(S_{ab}) &= (qT(S_{ca}))^*\bar{q} \\ &= (qT(\alpha_1))^*\bar{q} = T(\alpha_2). \end{aligned}$$

Let V' be the set of all nodes d in V such that d is on a path from $\mathfrak{c}$ to \$ which does not pass through the edge (a, q, c). Note that $\mathfrak{c}$ and \$ are in V'. Let $S' = (V', \Gamma', \varphi', \mathfrak{c})$, where Γ' and φ' are defined as follows.

(i) $\Gamma'(d) = \Gamma(d)$ for each d in $V' - \{a\}$, $\Gamma'(a) = b$;

(ii) $\varphi'(d) = \varphi(d)$ for each d in $V' - \{a\}$, $\varphi'(a) = f$,

where f is a new function symbol not occurring in S. Clearly, S' is a flowchart, and every section of S' is also a section of S. But W is not a section of S' since there is no nonnull path in S' going from a to c. Thus S' has fewer sections than S. By the induction hypothesis, there exists a while program α_3 such that $T(\alpha_3) = T(S')$. Let α be the result of replacing each occurrence of f in α_3 by α_2.

To complete the proof, it suffices to show that $T(S) = T(\alpha)$. Let $x = l_1 l_2 \cdots l_k$ be any string in $T(S)$. Then there is a path in S of the form

$$u = (a_0, l_1, a_1)(a_1, l_2, a_2) \cdots (a_{k-1}, l_k, a_k)$$

with $a_0 = \mathfrak{c}$ and $a_k = \$$. Suppose that u does not pass through a. Then u is also a path in S'. Thus x is in $T(S') = T(\alpha_3)$. Since $l_i \neq f$ for all i, x is in $T(\alpha)$. Now suppose that u passes through a. Let i be the smallest integer such that $a_i = a$. Let j be the largest integer such that $a_j = a$. Then $(a_j, l_{j+1}, a_{j+1}) = (a, \bar{q}, b)$. By construction,

$$(a_0, l_1, a_1) \cdots (a_{i-1}, l_i, a_i)(a, f, b)(a_{j+1}, l_{j+2}, a_{j+2}) \cdots (a_{k-1}, l_k, a_k)$$

is a path in S'. Thus $l_1 \cdots l_i f l_{j+2} \cdots l_k$ is in $T(\alpha_3)$. Moreover,

$$(a_i, l_{i+1}, a_{i+1}) \cdots (a_j, l_{j+1}, a_{j+1})$$

is a path in S_{ab}. Thus $l_{i+1} \cdots l_{j+1}$ is in $T(\alpha_2)$. Hence $l_1 \cdots l_i l_{i+1} \cdots l_{j+1} l_{j+2} \cdots l_k = x$ is in $T(\alpha)$. Thus, $T(S) \subset T(\alpha)$. Let x be in $T(\alpha)$. Since each element of $T(\alpha_3)$ contains at most one occurrence of f, either of the following hold.

(1) x is in $T(\alpha_3)$.

(2) There exist x_1, x_2 and x_3 such that $x = x_1x_2x_3$, x_1fx_3 is in $T(\alpha_3)$ and x_2 is in $T(\alpha_2)$.

Consider (1). Since $T(\alpha_3) = T(S')$, S' possesses a path u from ¢ to \$ spelling out x. Since x has no occurrence of f, u does not pass through the node a. Thus u is also a path in S (going from ¢ to \$). Hence x is in $T(S)$.

Consider (2). Let u be a path in S' from ¢ to \$ spelling out x_1fx_3. Then u is of the form $u_1(a, f, b)u_3$, where u_1 spells out x_1 and u_3 spells out x_3. Let u_2 be a path in S_{ab} from a to b spelling out x_2. Then $u_1u_2u_3$ is a path in S spelling out $x_1x_2x_3$. Thus $x = x_1x_2x_3$ is in $T(S)$. Thus $T(\alpha) \subset T(S)$, from which $T(\alpha) = T(S)$.

4. Minimal Flowcharts

In this section we show that, for any "minimal" flowchart S, if S is translatable, then S is modular.

Definition. Let $S = (V, \Gamma, \varphi, ¢)$ be a flowchart and \$ be the halt node of S. For each a in V, we shall denote by $T(S, a)$ the set of all strings x such that there exists a path from a to \$ which spells out x. Thus, $T(S, ¢) = T(S)$ and $T(S, \$) = \{\lambda\}$. For a and b in V, we write $a \equiv b$ if and only if $T(S, a) = T(S, b)$. $\equiv$ is obviously an equivalence relation on V.

A flowchart S is "minimal" if and only if $a \not\equiv b$ for all $a \neq b$ in V.

Lemma 4.1. *Given any flowchart $S = (V, \Gamma, \varphi, ¢)$, it is possible to find a minimal flowchart S' such that $T(S) = T(S')$.*

Proof. Let V' be the set of equivalence classes of $\equiv$ and $[a]$ the element of V' containing a. Define

$$\Gamma'([a]) = \begin{cases} ([b], [c]) & \text{if } \Gamma(a) = (b, c), \\ [b] & \text{if } \Gamma(a) = b, \\ \text{undefined} & \text{otherwise;} \end{cases}$$

$$\varphi'([a]) = \varphi(a).$$

Clearly, the definition is consistent. The flowchart $S' = (V', \Gamma', \varphi', [¢])$ is the one required.

LEMMA 4.2. *If S is a minimal flowchart and (a, b) is any linked pair of nodes in S, then the subflowchart S_{ab} determined by (a, b) is also minimal.*

Proof. Let c and d be nodes of S_{ab}. Suppose that $c \equiv d$ in S_{ab}. To prove the lemma it suffices to show that $c \equiv d$ in S. Let x be in $T(S, c)$. Then there exists a path

$$u = (a_0, l_1, a_1)(a_1, l_2, a_2) \cdots (a_{k-1}, l_k, a_k)$$

such that $a_0 = c$, $a_k = \$$ and $x = l_1 l_2 \cdots l_k$. Since (a, b) is linked, u passes through b. Let i be the smallest integer such that $a_i = b$. Then $l_1 l_2 \cdots l_i$ is in $T(S_{ab}, c)$. Since $T(S_{ab}, c) = T(S_{ab}, d)$, there exists a path v from d to b spelling out $l_1 l_2 \cdots l_i$. Then

$$v(a_i, l_{i+1}, a_{i+1}) \cdots (a_{k-1}, l_k, a_k)$$

is a path from d to \$ which spells out x. Hence x is in $T(S, d)$. Therefore $T(S, c) \subset T(S, d)$. An analogous argument shows that $T(S, d) \subset T(S, c)$. Thus we have $T(S, c) = T(S, d)$, that is, $c \equiv d$ in S.

DEFINITION. We say a string x is a "prefix" of a string y if there is a string z such that $y = xz$. If, moreover, $z \neq \lambda$, then x is a "proper" prefix. A set X of strings is said to have the "prefix property" if no proper prefix of a string in X is also in X. It follows that every trace set has the prefix property.

LEMMA 4.3. *Let S be a minimal flowchart with start node $\mathcal{C}$ and halt node \$. Let α be a while program of the form (while q do β)γ. If $T(S) = T(\alpha)$, then $(c, \mathcal{C})$ is linked, $T(S_{c\mathcal{C}}) = T(\beta)$ and $T(S_{b\$}) = T(\gamma)$, where $(\mathcal{C}, q, c)$ and $(\mathcal{C}, \bar{q}, b)$ are edges of S.*

Proof. Suppose that $T(S) = T(\alpha)$. Clearly $(\mathcal{C}, q, c)$ and $(\mathcal{C}, \bar{q}, b)$ are in $\tilde{S}$ for some c and b.

Let $u = (a_0, l_1, a_1) \cdots (a_{m-1}, l_m, a_m)$ be a path from c to \$. Since $ql_1 \cdots l_m$ is in $T(\alpha)$, $l_1 \cdots l_i$ is in $T(\beta)$ for some i. To show that $(c, \mathcal{C})$ is linked, it suffices to prove

$$a_i = \mathcal{C} \tag{1}$$

Let z be in $T(S, a_i)$. Then $ql_1 \cdots l_i z$ is in $T(S)$, and hence $T(\alpha)$. Since $l_1 \cdots l_i$ is in $T(\beta)$ and $T(\beta)$ has the prefix property, z is in $T(\alpha) = T(S)$. Thus $T(S, a_i) \subset T(S)$. Let z be in $T(S) = T(\alpha)$. Then $ql_1 \cdots l_i z$ is in $T(\alpha)$, and hence in $T(S)$. Since there is exactly one path from $\mathcal{C}$ which spells out $ql_1 \cdots l_i$, there must be a path from a_i to \$ which spells out z. Hence z is in $T(S, a_i)$. Therefore $T(S) = T(S, a_i)$. Since S is minimal, we have $a_i = \mathcal{C}$.

Obviously, $T(S_{b\$}) = T(\gamma)$. Thus we show that $T(S_{c\mathcal{C}}) = T(\beta)$.

Let y be the shortest string in $T(S_{b\$}) = T(\gamma)$. Then there exists a path u from b to \$ spelling out y. Let x be any string in $T(S_{c\not c})$. Then there exists a path

$$v = (a_0, l_1, a_1) \cdots (a_{k-1}, l_k, a_k)$$

such that $a_0 = c$, $a_i \neq \not c$ for $0 \leqslant i < k$, $a_k = \not c$ and $x = l_1 \cdots l_k$. (If $x = \lambda$, then by convention we put $v = \lambda$. Note that $x = \lambda$ if and only if $S_{c\not c} = \Lambda$.)

Since $(\not c, q, c)v(\not c, \bar{q}, b)u$ is a path in S going from $\not c$ to \$, $qx\bar{q}y$ is in $T(S)$. Since $T(S) = T(\alpha)$,

$$qx\bar{q}y = qx'x''\bar{q}y'$$

for some x' in $T(\beta)$, x'' in $(qT(\beta))^*$ and y' in $T(\gamma)$. Since $|y| \leqslant |y'|$, x' is a prefix of x (where $|y|$ denotes the length of y). Suppose $x = \lambda$. Then $x' = x = \lambda$. Thus x is in $T(\beta)$. Now suppose that $x \neq \lambda$. Let i be the integer such that $l_1 \cdots l_i = x'$. Since $(a_0, l_1, a_1) \cdots (a_{k-1}, l_k, a_k)(\not c, \bar{q}, b)u$ is a path from c to \$ and $l_1 \cdots l_i$ is in $T(\beta)$, it follows from Eq. (1) that $a_i = \not c$. Then $i = k$, so that $x = l_1 \cdots l_k = x'$. Hence x is in $T(\beta)$. Therefore $T(S_{c\not c}) \subset T(\beta)$.

To see the reverse inclusion, let x be a string in $T(\beta)$. Then $qx\bar{q}y$ is in $T(\alpha) = T(S)$. Thus

$$qx\bar{q}y = qx_1qx_2 \cdots qx_t\bar{q}y'$$

for some $x_1, \ldots, x_t$ in $T(S_{c\not c})$ and y' in $T(S_{b\$})$. Since $|y| \leqslant |y'|$, x_1 is a prefix of x. We already know that $T(S_{c\not c}) \subset T(\beta)$. Thus x_1 is in $T(\beta)$. Then, by the prefix property, $x_1 = x$. Thus x is in $T(S_{c\not c})$. Hence $T(\beta) \subset T(S_{c\not c})$, from which $T(\beta) = T(S_{c\not c})$.

DEFINITION. The "length" of a while program is defined as follows.

(1) $|\lambda| = 0$;

(2) $|f| = 1$, f in F;

(3) $|\alpha\beta| = |\alpha| + |\beta|$;

(4) $|(\textit{if } q \textit{ then } \alpha \textit{ else } \beta)| = 1 + |\alpha| + |\beta|$;

(5) $|(\textit{while } q \textit{ do } \alpha)| = 1 + |\alpha|$, where α and β are while programs.

THEOREM 4.4. *Let S be a minimal flowchart. If S is translatable, then S is modular.*

Proof. Let $S = (V, \Gamma, \varphi, \not c)$, and let \$ be the halt node of S. Let α be a while program such that $T(S) = T(\alpha)$. We proceed by induction on the length of α.

Suppose that $|\alpha| = 0$. Then $T(\alpha) = T(S) = \{\lambda\}$. Hence $S = \Lambda$. Thus S is modular.

Suppose that $|\alpha| > 0$. Three cases arise.

Case I. $\alpha = f\gamma$, with f in F. In this case, $\not c$ must be a function node. Let $b = \Gamma(\not c)$. Then $(\not c, f, b)$ is in $\tilde{S}$. Let $S_{b\$}$ be the subflowchart determined by $(b, \$)$. Clearly,

$T(S_{b\$}) = T(\gamma)$. By Lemma 4.2, $S_{b\$}$ is minimal. By the induction hypothesis, $S_{b\$}$ is modular. Let W be any section of S. Then, for any node d in W, there is a path from b to d. Thus W is a section of $S_{b\$}$. Since $S_{b\$}$ is modular, W has a unique exit. Therefore S is modular.

Case II. $\alpha = (\textit{if } q \textit{ then } \beta \textit{ else } \gamma)\delta$. Then $\not\subset$ must be a test node. Thus, $(\not\subset, \bar{q}, b)$ and $(\not\subset, q, c)$ are in $\tilde{S}$ for some b and c in V. Let $S_{b\$}$ and $S_{c\$}$ be the subflowcharts determined by $(b, \$)$ and $(c, \$)$, respectively. Clearly, $T(S_{b\$}) = T(\gamma\delta)$ and $T(S_{c\$}) = T(\beta\delta)$. By Lemma 4.2, both $S_{b\$}$ and $S_{c\$}$ are minimal. Thus, by the induction hypothesis, $S_{b\$}$ and $S_{c\$}$ are modular. Let W be any section of S. Clearly, $W - \{\not\subset\} \neq \varnothing$. Let d be in $W - \{\not\subset\}$. Then, either $S_{b\$}$ or $S_{c\$}$ contains d. Suppose that $S_{b\$}$ contains d. (An analogous argument holds if $S_{c\$}$ contains d.) Then $S_{b\$}$ contains all nodes of W. Hence W is a section of $S_{b\$}$. Since $S_{b\$}$ is modular, W has exactly one exit.

Case III. $\alpha = (\textit{while } q \textit{ do } \beta)\gamma$. Then $(\not\subset, q, c)$ and $(\not\subset, \bar{q}, b)$ are in $\tilde{S}$ for some c and b in V. By Lemma 4.3, $T(S_{c\not\subset}) = T(\beta)$ and $T(S_{b\$}) = T(\gamma)$. By Lemma 4.2, $S_{b\$}$ and $S_{c\not\subset}$ are minimal. By the induction hypothesis, $S_{b\$}$ and $S_{c\not\subset}$ are modular.

Let W be any section of S. An argument analogous to Case II shows that W is either a section of $S_{b\$}$ or a section of $S_{c\not\subset}$. Thus W has a unique exit.

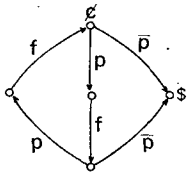
Combining Theorem 3.3 and 4.4, we have the following main result of this paper.

THEOREM 4.5. *Let S be a minimal flowchart. Then S is translatable if and only if S is modular.*

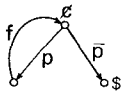
Algorithm. Let S be a flowchart. To determine whether S is translatable,

(1) Find a minimal flowchart S' congruent to S using Lemma 4.1,

(2) If S' is modular, then S is translatable. Otherwise S is not translatable.



(a)



(b)

FIG. 3. An example of the algorithm.

Remark. Theorem 4.4 cannot be extended to an arbitrary flowchart. For example, consider the flowchart S of Fig. 3(a). The flowchart S' of Fig. 3(b) is a minimal flowchart congruent to S. Since S' is modular, S' is translatable, and hence so is S. However, S is not modular.

5. Regular Collapsibility

In this section we introduce a stronger version of the "collapsibility" of Hecht and Ullman [7], called regular collapsibility, and show it equivalent to modularity.

Transformation A. Let $S = (v, \Gamma, \varphi, \not\epsilon)$ be a flowchart, and let a and b be nodes in S such that b is the unique proper successor of a. Let $E = \{(c, l, a) \text{ in } \tilde{S} \mid c \neq a\}$. From S, we construct a flowchart S' as follows.

(1) Delete the node a and delete all edges incident out from a.

(2) Delete all edges in E and add an edge (c, l, b) for each (c, l, a) in E.

(3) If $a = \not\epsilon$, then b is the start node of S'. Otherwise $\not\epsilon$ is the start node of S'.

From the definition of flowcharts, we immediately have the following.

Lemma 5.1. *Let $S = (V, \Gamma, \varphi, \not\epsilon)$ be a flowchart. Then a node a has a unique proper successor if and only if one of the following conditions holds.*

(1) *a is a function node;*

(2) *a is a test node and $\Gamma_1(a) = \Gamma_2(a)$;*

(3) *a is a test node and $a = \Gamma_i(a)$ for some i.*

Proof. Notice that every function or test node has at least one proper successor. In the view of above, the transformation A is equivalent to the following transformation B.

Transformation B. Let Q_1, Q_2 and Q_3 be flowcharts in Fig. 4, where b is a successor of a, f is in F, q and $\bar{q}$ are in $\hat{P}$. If S contains one of Q_1 to Q_3 as a subgraph, then replace it by a new node, say d. Thus, there is an edge of the form (d, l, c), $c \neq a$, $c \neq b$, if there was previously an edge (b, l, c). There is an edge of the form (c, l, d), $c \neq a$, $c \neq b$, if there was previously either (c, l, a) or (c, l, b).

Definition. Let S and S' be flowcharts. We write $S \Rightarrow S'$ if and only if S can be transformed into S' by the transformation A. If A is applied to node a in S to yield S', then we sometimes write $S \underset{a}{\Rightarrow} S'$. For S and S' we write $S \overset{*}{\Rightarrow} S$ if either $S = S'$, or

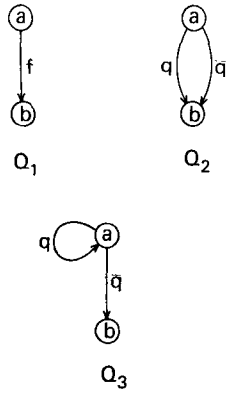


FIG. 4. Subflowcharts Q_1, Q_2 and Q_3.

there exist S_0, S_1,..., S_k such that $S = S_0$, $S_k = S'$ and $S_i \Rightarrow S_{i+1}$ for each i. The sequence S_0, S_1,..., S_k is called a "reduction" (of length k) and is denoted by

$$S_0 \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_k .$$

If $S \overset{*}{\Rightarrow} S'$ and there is no S'' such that $S' \Rightarrow S''$, then we write $S \hat{\Rightarrow} S'$.

DEFINITION. A flowchart S is said to be "regularly collapsible" if and only if $S \overset{*}{\Rightarrow} \Lambda$.

EXAMPLE. The flowchart S of Fig. 1(a) is regularly collapsible. A typical reduction is in Fig. 5. On the other hand, the flowchart of Fig. 1(b) is not regularly collapsible, since no node has a unique proper successor.

We next show that the relation $\hat{\Rightarrow}$ is a function.

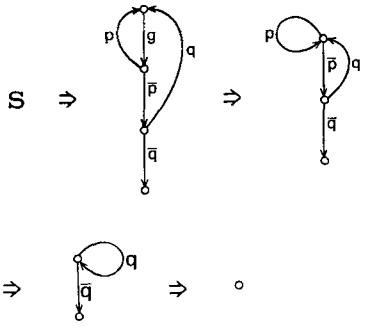


FIG. 5. Reduction for the flowchart S of Fig. 1(a).

LEMMA 5.2. *$S \hat{\Rightarrow} S'$ and $S \hat{\Rightarrow} S''$ implies $S' = S''$.*

Proof. The proof will proceed by induction on the number of nodes of S. Note that each application of the transformation A deletes exactly one node. If $S = \Lambda$, then $S' = S'' = \Lambda$, and the lemma is obviously true.

Suppose that S has n nodes, $n > 1$, and that the lemma is true for all flowcharts with fewer than n nodes. Let

$$S \underset{a}{\Rightarrow} S_1 \hat{\Rightarrow} S', \tag{1}$$

$$S \underset{b}{\Rightarrow} S_2 \hat{\Rightarrow} S''. \tag{2}$$

If $a = b$, then $S_1 = S_2$. Thus $S' = S''$ by the induction hypothesis. If $a \neq b$, then A may be performed on b in S_1 and on a in S_2 to yield the same flowchart, say S_3. Let S''' be a flowchart such that $S_3 \hat{\Rightarrow} S'''$, and consider the following reductions.

$$S_1 \underset{b}{\Rightarrow} S_3 \hat{\Rightarrow} S''', \tag{3}$$

$$S_2 \underset{a}{\Rightarrow} S_3 \hat{\Rightarrow} S'''. \tag{4}$$

By Eqs. (1) and (3), $S' = S'''$. By Eqs. (2) and (4), $S'' = S'''$. Thus $S' = S''$.

DEFINITION. Let $S = (V, \Gamma, \varphi, \not\subset)$ be a flowchart, and $Q = (V', \Gamma', \varphi', \not\subset')$ be a subflowchart of S. Let b be the halt node of Q. Let $E = \{(c, l, d) \text{ in } \tilde{S} \mid c \text{ in } V - V_{\$}', d \text{ in } V_{\$}'\}$, where $V_{\$}' = V' - \{b\}$. From S, we construct a flowchart S/Q, called the "quotient," as follows.

(1) Delete all nodes in $V_{\$}'$ and delete all edges incident out from $V_{\$}'$.

(2) Delete all edges in E and add an edge (c, l, b) for each (c, l, d) in E.

(3) If $\not\subset$ is in $V_{\$}'$, then b is the start node of S/Q. Otherwise $\not\subset$ is the start node of S/Q.

LEMMA 5.3. *Let Q be a subflowchart of S. If $Q \overset{*}{\Rightarrow} \Lambda$, then $S \overset{*}{\Rightarrow} S/Q$.*

Proof. The proof will be by induction on the number of nodes in Q. If $Q = \Lambda$, then $S = S/Q$ and the lemma is obviously true. Now suppose that Q has n nodes, $n > 1$. Let b be the halt node of Q, and let

$$Q \underset{a}{\Rightarrow} Q_1 \overset{*}{\Rightarrow} \Lambda, \qquad \text{and} \qquad S \underset{a}{\Rightarrow} S_1.$$

Note that we cannot have $a = b$, since b has no successor in Q. Clearly Q_1 is a subflowchart of S_1. By the induction hypothesis, $S_1 \overset{*}{\Rightarrow} S_1/Q_1$. To complete the proof,

it suffices to show that $S/Q = S_1/Q_1$. Let $S/Q = (V, \Gamma, \varphi, \phi)$ and $S_1/Q_1 = (V', \Gamma', \varphi', \phi')$. Let ϕ_0 be the start node of S.

We now show that $\phi = \phi'$. Suppose that Q contains ϕ_0. Then Q_1 contains the start node of S_1. Thus, $\phi = \phi' = b$. Suppose that Q does not contain ϕ_0. Then ϕ_0 is the start node of S_1, and Q_1 does not contain ϕ_0. Thus, $\phi = \phi' = \phi_0$. In either case, we have $\phi = \phi'$.

We now prove that $V = V'$, $\Gamma = \Gamma'$ and $\varphi = \varphi'$. To do this, it suffices to show that $\widetilde{S/Q} = \widetilde{S_1/Q_1}$. By definition,

$$
\begin{aligned}
& (c, l, b) \text{ is in } \widetilde{S/Q} \\
\text{iff } & (c, l, b_1) \text{ is in } \tilde{S} - \tilde{Q} \text{ for some } b_1 \text{ in } Q \\
\text{iff } & (c, l, b_2) \text{ is in } \tilde{S}_1 - \tilde{Q}_1 \text{ for some } b_2 \text{ in } Q_1 \\
\text{iff } & (c, l, b) \text{ is in } \widetilde{S_1/Q_1}.
\end{aligned}
$$

$$
\begin{aligned}
& (c, l, d) \text{ is in } \widetilde{S/Q} \text{ and } d \neq b \\
\text{iff } & (c, l, d) \text{ is in } \tilde{S} - \tilde{Q} \text{ and } d \text{ is not in } Q \\
\text{iff } & (c, l, d) \text{ is in } \tilde{S}_1 - \tilde{Q}_1 \text{ and } d \text{ is not in } Q_1 \\
\text{iff } & (c, l, d) \text{ is in } \widetilde{S_1/Q_1} \text{ and } d \neq b.
\end{aligned}
$$

Hence $\widetilde{S/Q} = \widetilde{S_1/Q_1}$.

LEMMA 5.4. *Let $S \Rightarrow S'$. Then S is modular if and only if S' is modular.*

Proof. Let $S \underset{a}{\Rightarrow} S'$. Let b be the unique proper successor of a.

(I) Suppose that S contains a section W with more than one exit. Let (c_1, l_1, d_1) and (c_2, l_2, d_2) be two distinct exits of W. For $i = 1$ and 2, let

$$u_i = e_1^i e_2^i \cdots e_{k(i)}^i, \qquad e_j^i \quad \text{in} \quad \tilde{S},$$

be a path from d_i to the halt node which does not pass through W. We can not have $c_1 = c_2$, for this would imply that neither d_1 nor d_2 is in W (this contradicts Lemma 3.1).

Now suppose that W contains a. Suppose $a = c_i$. Then $b = d_i$. Since b is the unique proper successor of a and d_i is not in W, it follows that $W = \{a\}$. This is a contradiction, since W contains at least two distinct nodes, c_1 and c_2. Thus $a \neq c_1$ and $a \neq c_2$. Since any W-path from a to c_1 passes through the node b, b is in W. Now we show that

$$W' = W - \{a\} \text{ is a section of } S'. \tag{1}$$

Let a' and b' be arbitrary nodes of W'. Then there is a W-path

$$v = (a_0 , l_1 , a_1)(a_1 , l_2 , a_2) \cdots (a_{k-1} , l_k , a_k)$$

in S, with $a_0 = a'$ and $a_k = b'$. If v does not pass through the node a, then v is also a W'-path in S'. Suppose that v passes through a. We may assume that v is elementary. (Otherwise, we can find a elementary W-path going from a' to b'.) Let $a_i = a$. Then $0 < i < k$ and $a_{i+1} = b$. Since $(a_{i-1} , l_i , a_{i+1})$ is an edge of S',

$$(a_0 , l_1 , a_1) \cdots (a_{i-2} , l_{i-1} , a_{i-1})(a_{i-1} , l_i , a_{i+1})(a_{i+1} , l_{i+2} , a_{i+2}) \cdots (a_{k-1} , l_k , a_k)$$

is a W'-path of S'. Thus (1) is proved.

Since neither u_1 nor u_2 contains the node a, both of them are paths of S'. Since neither u_1 nor u_2 passes through W', (c_1 , l_1 , d_1) and (c_2 , l_2 , d_2) are exits of W'. Thus S' is not modular.

Now suppose that W does not contain a. Let $W' = W$. Clearly W' is a section of S'. For each i, $i = 1, 2$, let

$$d_i' = \begin{cases} b & \text{if } \ d_i = a, \\ d_i & \text{otherwise.} \end{cases}$$

To complete the proof of (I) it suffices to show that

$$(c_1 , l_1 , d_1') \quad \text{and} \quad (c_2 , l_2 , d_2') \quad \text{are exits of} \quad W'. \tag{2}$$

Clearly (c_1 , l_1 , d_1') and (c_2 , l_2 , d_2') are edges of S'. Suppose that u_i does not contain a. Then u_i is also a path in S' going from d_i' to the halt node. Since u_i does not pass through W, (c_i , l_i , d_i') is an exit of W'. Suppose that u_i contains a. We may assume that u_i is elementary. If the initial node of ${e_1}^i$ is a, then $d_i' = b$ and ${e_2}^i {e_3}^i \cdots e_{k(i)}^i$ is a path in S' going from b to the halt node. Suppose that the initial node of ${e_j}^i$, $j \neq 1$, is a. Then $e_{j-1}^i = (a', l', a)$ for some a' and l', and ${e_j}^i = (a, l'', b)$ for some l''. Then

$${e_1}^i \cdots e_{j-2}^i (a', l', b)\, e_{j+1}^i \cdots e_{k(i)}^i$$

is a path in S' (going from d_i' to the halt node) which does not pass through W'.

(II) Suppose that S' contains a section W' with more than one exit. Let (c_1 , l_1 , d_1') and (c_2 , l_2 , d_2') be two distinct exits of W'. For each i, $i = 1, 2$, let u_i' be a path in S' going from d_i' to the halt node which does not pass through W'. We first note that for all (c, l, d) in $\tilde{S}'$

$$\text{if } (c, l, d) \text{ is not in } \tilde{S}, \text{ then } d = b \text{ and } (c, l, a) \text{ is in } \tilde{S}. \tag{3}$$

Clearly S contains an edge of the form (a, l', b). Let μ be the (semigroup) homomorphism from $(\tilde{S}')^*$ into $(\tilde{S})^*$ defined by

$$\mu(c, l, d) = \begin{cases} (c, l, d) & \text{if } (c, l, d) \text{ is in } \tilde{S}, \\ (c, l, a)(a, l', b) & \text{otherwise.} \end{cases}$$

Then $\mu(v)$ is a path of S for every path v of S'.

Suppose that W' is not a section of S. Let $W = W' \cup \{a\}$. We now show that W is a section of S. Since W' is not a section of S, there exist (c, l, d) in $\tilde{S}'$, c and d in W', such that (c, l, d) is not in $\tilde{S}$. By (3), $d = b$ and (c, l, a) is in $\tilde{S}$. Thus b is in W'. Let a' be an arbitrary element of W'. Let v_1 be a W'-path of S' going from a' to c. Then $\mu(v_1)(c, l, a)$ is a W-path of S going from a' to a. Let v_2 be a W'-path of S' going from b to a'. Then $(a, l', b)\,\mu(v_2)$ is a W-path of S going from a to a'. Thus W is a section of S. Since neither u_1' nor u_2' contains the node b, both of them are paths of S. Thus (c_1, l_1, d_1') and (c_2, l_2, d_2') are exits of W. Hence S is not modular.

Now suppose that W' is a section of S. For each i, $i = 1, 2$, let

$$(c_i, l_i, d_i) = \begin{cases} (c_i, l_i, d_i') & \text{if } (c_i, l_i, d_i') \text{ is in } \tilde{S}, \\ (c_i, l_i, a) & \text{otherwise.} \end{cases}$$

An analogous argument shows that (c_1, l_1, d_1) and (c_2, l_2, d_2) are exist of W' in S. Thus S is not modular.

COROLLARY 5.5. *If a flowchart S is regularly collapsible, then S is modular.*

Proof. Assume, for contradiction, that S is not modular and $S \overset{*}{\Rightarrow} \Lambda$. Then, by Lemma 5.4, Λ is not modular, that is, Λ contains a section with more than one exit. This is impossible.

THEOREM 5.6. *A flowchart S is regularly collapsible if and only if S is modular.*

Proof. By Corollary 5.5, it suffices to show the "if" portion. We do this by induction on the number of sections of S. If there is no section in S, then S contains no loop and can be transformed into Λ by repeated application of the transformation B on Q_1 and Q_2 in Fig. 4. Suppose that S is a modular flowchart with n sections, $n \geqslant 1$. Let W be a maximal section of S. Then W has exactly one exit $(a, \bar{q}, b)$, $\bar{q}$ in $\hat{P}$. Let (a, q, c) be the associate of $(a, \bar{q}, b)$. By Lemma 3.2, both (a, b) and (c, a) are linked. Let S_{ab} and S_{ca} be the subflowchart determined by (a, b) and (c, a), respectively. Then S_{ab} is of the form of Fig. 2. Clearly, S_{ca} is modular. Also, S_{ca} has fewer sections than S. Thus, by the induction hypothesis, $S_{ca} \overset{*}{\Rightarrow} \Lambda$. By Lemma 5.3, we have

$$S_{ab} \overset{*}{\Rightarrow} S_{ab}/S_{ca} = Q_3 \Rightarrow \Lambda.$$

Thus, by Lemma 5.3, $S \overset{*}{\Rightarrow} S/S_{ab}$. By Lemma 5.4, S/S_{ab} is modular. By Lemma 3.2, S/S_{ab} has fewer sections than S. Thus $S/S_{ab} \overset{*}{\Rightarrow} \Lambda$. Hence $S \overset{*}{\Rightarrow} \Lambda$.

EXAMPLE (Knuth and Floyd [10]). Consider the following program.

for i: $= 1$ *step* 1 *until* n *do*
if $A[i] = x$ *then go to* found;
not found: n: $= i$; $A[i]$: $= x$; $B[i]$: $= 0$;
found: $B[i]$: $= B[i] + 1$;

The flowchart S of this program is in Fig. 6. Then $S \hat{\Rightarrow} S'$, where S' is the flowchart of Fig. 1(b). Thus S is not regularly collapsible. Hence S can not be translated into while programs.

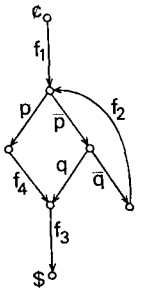


FIG. 6. An example of a flowchart which is not translatable into any while program. $p \equiv i > n$?. $q \equiv A[i] = x$?. $f_1 \equiv i := 1$. $f_2 \equiv i := i + 1$. $f_3 \equiv B[i] := B[i] + 1$. $f_4 \equiv n := 1$; $A[i] := x$; $B[i] := 0$.

ACKNOWLEDGMENTS

The author wishes to express deep appreciation to the referee of this paper for his careful reading of the manuscript and his many helpful suggestions about the style of presentation. The author is also indebted to Professor Satoru Takasu for his advice and suggestions toward this paper. He also wishes to thank Miss S. Nio for her efficient work in typing the manuscript.

REFERENCES

1. J. BRUNO AND K. STEIGLITZ, "The expression of algorithms by charts," TR 88, Computer Sciences Laboratory, Princeton University, Princeton, NJ, 1971.
2. D. C. COOPER, Some transformations and standard forms of graphs with application to computer programs, *in* "Machine Intelligence," Vol. 2, pp. 21–32, American Elsevier, New York, 1968.

3. D. C. COOPER, Programs for mechanical program verification, *in* "Machine Intelligence," Vol. 6, pp. 43–62, American Elsevier, New York, 1971.
4. D. C. COOPER, Böhm and Jacopini's reduction of flow charts, *Comm. ACM* **10** (1967), 463, 473.
5. E. W. DIJKSTRA, GOTO statement considered harmful, *Comm. ACM* **11** (1968), 147–148.
6. E. ENGELER, Structure and meanings of elementary programs, *in* "Symposium on Semantics of Algorithmic Languages" (E. Engeler, Ed.), pp. 89–101, Springer–Verlag, New York, 1971.
7. M. S. HECHT AND J. D. ULLMAN, Flow graph reducibility, *SIAM J. Comput.* **1** (1972), 188–202.
8. C. A. R. HOARE, An axiomatic basis of computer programming, *Comm. ACM* **12** (1969), 576–580, 583.
9. C. A. R. HOARE, Procedures and parameters: an axiomatic approach, *in* "Lecture Notes in Mathematics 188" (E. Engeler, Ed.), pp. 102–116, Springer–Verlag, New York, 1970.
10. D. E. KNUTH AND R. W. FLOYD, Notes on avoiding GOTO statements, *Information Processing Letters* **1** (1971), 23–31.
11. Z. MANNA, "Introduction to Mathematical Theory of Computations," to be published.
12. J. R. RICE, The GOTO statement reconsidered, *Comm. ACM* **11** (1968), 538.